Le point `.` représente une seule instance de n'importe quel caractère sauf le caractère de fin de ligne.

Ainsi l'expression `t.c` représente toutes les combinaisons de trois lettres commençant par t et finissant par c, comme *tic*, *tac*, *tqc* ou *t9c*, alors que `b.l..` pourrait représenter par exemple *bulle*, *balai* ou *bêler*.

La paire de crochets `[ ]` représente une occurrence quelconque des caractères qu'elle contient.

Par exemple `[aeiouy]` représente une voyelle, et `Duran[dt]` désigne *Durand* ou *Durant*.

Entre les crochets, on peut noter un intervalle en utilisant le tiret[1]. Ainsi, `[0-9]` représente les chiffres de 0 à 9, et `[a-zA-Z]` représente une lettre minuscule ou majuscule.

On peut de plus utiliser l'accent circonflexe en première position dans les crochets pour indiquer « le contraire de ». Par exemple `[^a-z]` représente autre chose qu'une lettre minuscule, et `[^'"]` n'est ni une apostrophe ni un guillemet.

L'astérisque `*` est un *quantificateur*, il signifie aucune ou une ou plusieurs occurrences du caractère ou de l'élément qui le précède immédiatement.

L'expression `ab*` signifie la lettre a suivie de zéro ou plusieurs lettres b, par exemple *ab*, *abbb* et `[A-Z]*` correspond à zéro, une ou plusieurs lettres majuscules.

L'accent circonflexe `^` est une *ancre*. Il indique que l'expression qui le suit se trouve en début de ligne.

L'expression `^Depuis` indique que l'on recherche les lignes commençant par le mot *Depuis*.

Le symbole dollar `$` est aussi une *ancre*. Il indique que l'expression qui le précède se trouve en fin de ligne.

L'expression `suivante !$` indique que l'on recherche les lignes se terminant par *suivante !*

L'expression `^Les expressions régulières$` extrait les lignes ne contenant que la chaîne *Les expressions régulières*, alors que `^$` extrait les lignes vides.

La contre-oblique (ou **antislash**) `\` permet d'*échapper*[2] à la signification des métacaractères.

Ainsi `\.` désigne un véritable point, `\*` un astérisque, `\^` un accent circonflexe, `\$` un dollar et `\\` une contre-oblique.

Les expressions régulières étendues

Elles ajoutent cinq symboles qui ont les significations suivantes :

La paire de parenthèses `( )` est utilisée à la fois pour former des sous-motifs et pour délimiter des sous-expressions, ce qui permettra d'extraire des parties d'une chaîne de caractères.

L'expression `(to)*` désignera *to*, *tototo*, etc.

Le signe plus `+` est un *quantificateur* comme *, mais il signifie une ou plusieurs occurrences du caractère ou de l'élément qui le précède immédiatement.

L'expression `ab+` signifie la lettre a suivie d'une ou plusieurs lettres b.

1. Les caractères considérés dans cet intervalle sont ceux dont le code est compris entre les codes des deux caractères délimitatifs.

2. Les informaticiens utilisent fréquemment l'expression « échapper un caractère » pour « le préfixer par le caractère contre-oblique ». Par exemple, échapper n pour \n.

Le point d'interrogation `?`, troisième *quantificateur*, signifie zéro ou une instance de l'expression qui le précède.

Par exemple `écrans?` désigne *écran* ou *écrans*.

La paire d'accolades `{ }` nombre d'occurrences permises pour le motif qui le précède.

Par exemple `[0-9]{2,5}` attend entre deux et cinq chiffres décimaux.

Les variantes suivantes sont disponibles : `[0-9]{2,}` signifie au minimum deux occurrences de chiffres décimaux et `[0-9]{2}` deux occurrences exactement.

La barre verticale `|` représente des choix multiples dans un sous-motif.

L'expression `Duran[dt]` peut aussi s'écrire `(Durand|Durant)`.

On pourrait utiliser l'expression `(lu|ma|me|je|ve|sa|di)` dans l'écriture d'une date.

Dans de nombreux outils et langages (dont Python), la syntaxe étendue comprend aussi une série de séquences d'échappement permettant d'identifier des classes entières de caractères [1] :

Séquence	Signification
`\`	symbole d'échappement
`\e`	séquence de contrôle *escape*
`\f`	saut de page
`\n`	fin de ligne
`\r`	retour chariot
`\t`	tabulation horizontale
`\v`	tabulation verticale
`\d`	classe des chiffres
`\s`	classe des caractères d'espacement
`\w`	classe des caractères alphanumériques
`\b`	localisation de début ou de fin de mot
`\D`	négation de la classe `\d`
`\S`	négation de la classe `\s`
`\W`	négation de la classe `\w`
`\B`	négation de la classe `\b`

TABLEAU C.1 – Séquences d'échappement

Remarque

Par « caractères d'espacement » on entend espace, tabulation et retour à la ligne. Les caractères alphanumériques forment l'ensemble `[a-zA-Z0-9_]`.

Les expressions régulières avec Python

Toutes les expressions régulières de base, étendues et certaines expressions avancées (par exemple la capture des groupements, les groupements nommés, les options de compilation...) sont fournies par le module `re`. Les scripts Python devront donc comporter la ligne :

```
import re
```

1. Le standard Unicode définit ces classifications de caractères.

Pythonismes

Le module `re` fournit des outils utilisant la programmation objet. Les motifs et les correspondances de recherche seront des objets de la classe `SRE_Pattern` auxquels on pourra appliquer des méthodes.

Utilisation des *raw strings*

La syntaxe des motifs comprend souvent le caractère contre-oblique, qui doit être lui-même échappé dans une chaîne de caractères, ce qui alourdit la notation. On peut éviter cet inconvénient en utilisant des « chaînes brutes » préfixées par `r`. Par exemple au lieu de :

```
"\\d\\d? \\w+ \\d{4}"
```

on écrira :

```
r"\d\d? \w+ \d{4}"
```

Les options de compilation

Grâce à un jeu d'*options de compilation* des expressions, il est possible de piloter le comportement des expressions régulières. On utilise pour cela soit des paramètres supplémentaires au constructeur, soit plus fréquemment la syntaxe `(?<drapeau>)` avec les drapeaux suivants :

- **a** pour la correspondance alphanumérique restreinte à l'ASCII (Unicode par défaut) ;
- **i** pour la correspondance alphabétique non sensible à la casse ;
- **L** pour que les correspondances utilisent la *locale*, c'est-à-dire les particularités du pays ;
- **m** appliqué aux chaînes de plusieurs lignes ;
- **s** pour la modification du comportement du métacaractère point `.`, qui représentera alors aussi le saut de ligne ;
- **u** pour la correspondance alphanumérique Unicode (par défaut) ;
- **x** pour le mode « verbeux » (permet d'introduire des commentaires).

Voici un exemple de recherche non sensible à la casse :

```
>>> import re
>>> case = re.compile(r"[a-z]+")          # Là on est sensible à la casse
>>> print(case.search("Bastille").group())
astille
>>> ignore_case = re.compile(r"(?i)[a-z]+")    # Là non : utilisation du drapeau (?i)
>>> print(ignore_case.search("Bastille").group())
Bastille
```

Les motifs nominatifs

Python possède une syntaxe qui permet de nommer des parties de motif délimitées par des parenthèses, ce qu'on appelle un « motif nominatif » :

- syntaxe de création d'un motif nominatif : `(?P<nom_du_motif>)` ;
- syntaxe permettant de s'y référer : `(?P=nom_du_motif)` ;
- syntaxe à utiliser dans un motif de remplacement ou de substitution : `\g<nom_du_motif>`.